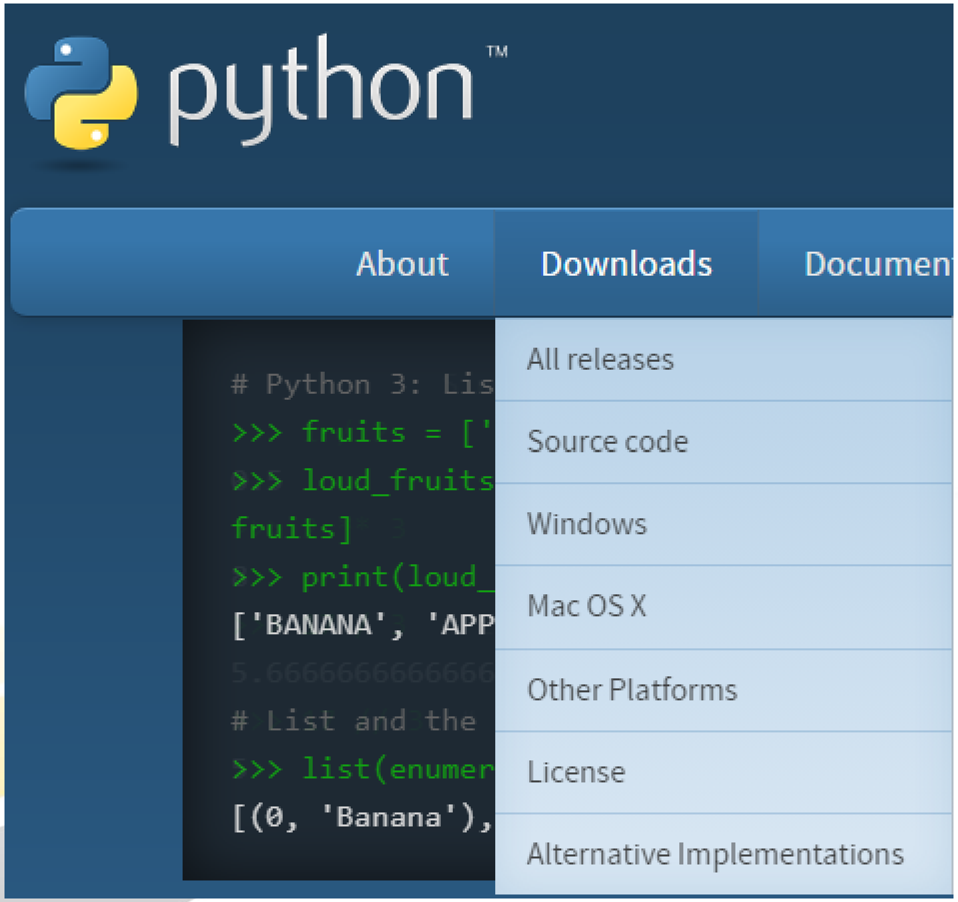
Python

Introduction to Programming

If you haven't installed Python 3

- On your laptop
- Go to python.org
- Download the latest version for your device



```
# Python 3: Lis
>>> fruits = ['
>>> loud_fruits
fruits]
>>> print(loud_
['BANANA', 'APP
5.666666666666
# List and the
>>> list(enumer
[(0, 'Banana'),
```

Python Programming Language

- Invented by Guido van Rossum
- Name after Monty Python (British Comedy Act)

Why Learn Python?

- It is easy to learn
- One downloadable app and you can start
- Uses a simple editor
- No Development Environment to learn
- Can easily add features (extensible)
- Use on any device with any OS (portable)
- Python is used in Data Science and AI
- There is a LOT of support out there (big Python community)

Python Programming Language

- General Purpose
  - Can be used for just about anything
  - Useable everywhere
  - Cross platform (Windows, Mac, Linux, Unix …)
- High level
  - Easier to learn
  - More “natural” concepts
  - Error handling
- Extensible
  - Easy to add features
  - Add features only as necessary
- Scalable
  - From tiny programs to chrome extensions to full applications

The Zen of Python

Python Guiding Principles

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Errors should never pass silently
- Readability counts

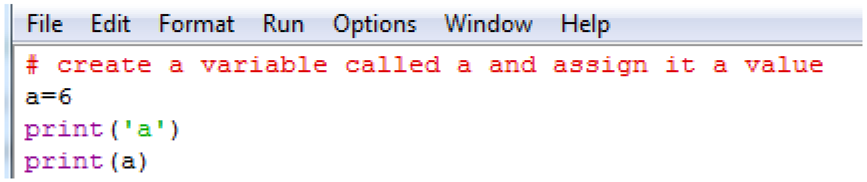
and more …

By Tim Peters, 2004

Readability Counts

- Your code should be easily readable by someone else
- Ideal for team programming
- Use lots of white space
- Well commented

Idle – the Python Editor



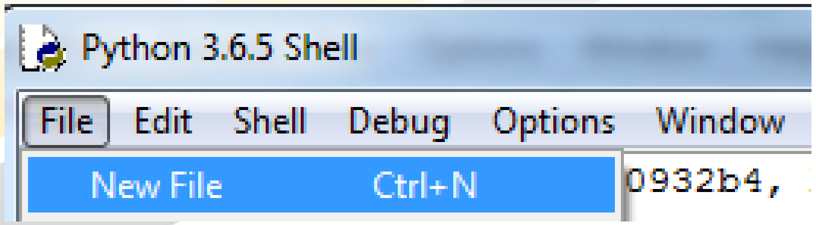
```
# create a variable called a and assign it a value
a=6
print('a')
print(a)
```

- Simple text editor where you write your code
- Named after Eric Idle of Monty Python
- Use of colour makes editing easier

- Sample code in Idle

Note

If you are in the Shell window, choose File->New File to open Idle

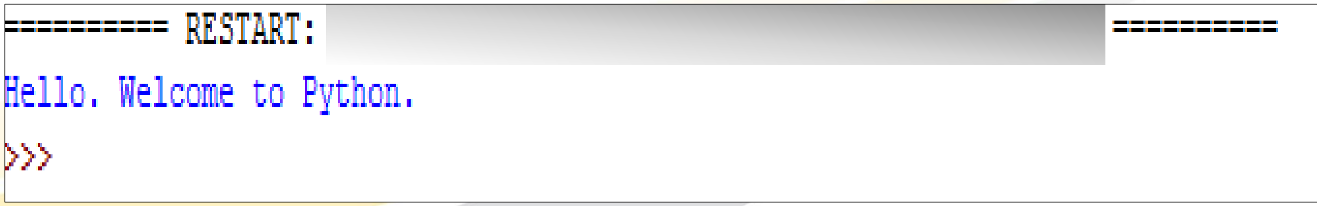


Your First Line of Code

- In Idle, type this code
  - `print('Hello. Welcome to Python')`

  Note how Python colours your code
- Save as `hello_welcome.py` (no need to type the `.py`)
- Choose run -> run module or click F5

In the shell window you should see this



```
========== RESTART:                                         ==========
Hello. Welcome to Python.
>>>
```

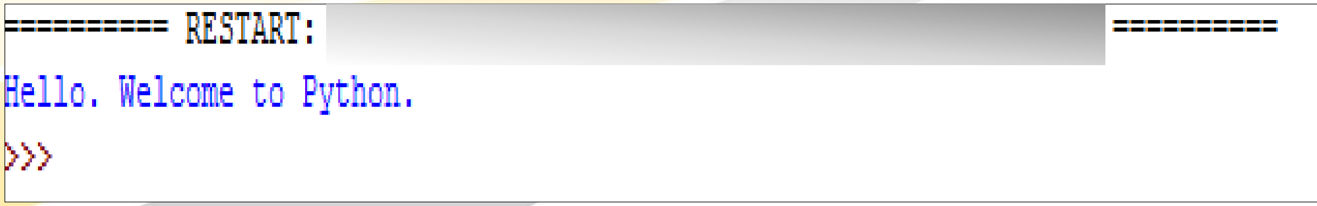
Add a Comment

- Leave the second line blank (think readability)
- On the third line type

```
# create a variable called num and assign it a value
```

- You have just typed your first comment.
- The # at the beginning marks this line as a comment.
- If you run this code, Python will ignore your comment
- Comments are to help you and other programmers

In the shell window you should see this

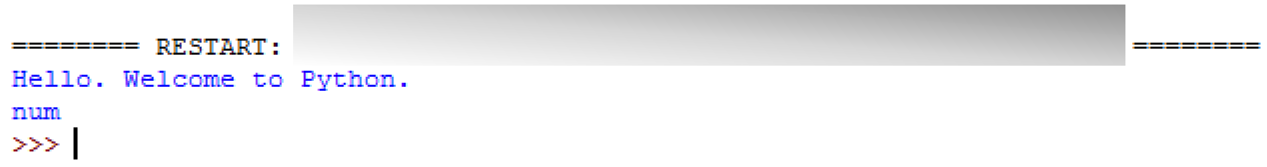


```
========== RESTART: [redacted] ==========
Hello. Welcome to Python.
>>>
```

Add more code - Comment

- Add another line of code
  - `print('num')`

  Note again how Python colours your code
- Run your code

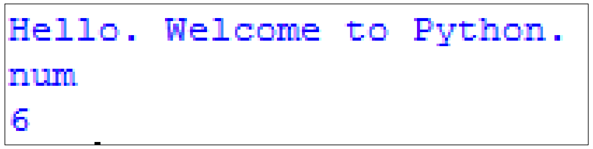


```
======== RESTART: [redacted] ========
Hello. Welcome to Python.
num
>>> |
```

- Python has printed the word num
- The technical term is a **`string`**
- It refers to letters or something in quotes.

Variables: Name, Value and Type

- Add two more lines of code
  - `num=6`
  - `print(num)`
- Run your code.



```
Hello. Welcome to Python.
num
6
```

- `num` is a **`variable`**
- The **`name`** of the variable is `num`
- The **`value`** of `num` is 6
- The **type** of `num` is `int` (integer)
- Note that **num** is not in quotes as it is a number not a string

Variable Names

Variable names in Python

- Must start with a letter
  - `user1` is valid variable name
  - `1user` is **not** a valid variable name
- May not have spaces
  - `user 1` is **not** a valid variable name
- Often contain underscores instead of spaces
  - Eg `user_1`
- Should be descriptive of what they relate to
  - Eg. A variable for a user's name should be `user_name` rather than `un` or `u_n`

The last two are typical of Python

Python Key Words (Reserved Words)

- Cannot be used as variable names

False	and	def	global	or
None	as	del	if	pass
True	assert	elif	import	raise
	async	else	in	return
	await	except	is	try
	break	finally	lambda	while
	class	for	nonlocal	with
	continue	from	not	yield

Variable Types

Python has two basic variable types

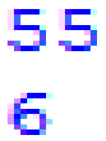
- String
  - Words or letters
  - Must uses quotes
  - Eg `user_name = 'Li'`
- Number
  - Must be a number
  - Do not use quotes
  - Eg `start_number = 6`

Numbers can be

- Integers
- Floats (decimals) eg `average_score = 69.25`

A Second Variable

- Add this code after `num=6`
  - `new_num=num`
  - `num=55`
  - `print(num)`
  - `print(new_num)`



```
55
6
```

- Run your code.

- What has happened?
- First the value of `num` is 6
- Then the value of `new_num` is also 6
- Then we change the value of `num` to 55
- So the value of `num` is 55 and the value of `new_num` is 6 (not changed).

Python Arithmetic Operators

- `+` (plus)
- `–` (minus)
- `*` (times, multiply)

- `/` (divide)
- `%` (mod)
- ** (power)

Examples

`a=6`

- `n=a+3` Now n is 9.
- `n=a-4` Now n is 2.
- `n=3*a` Now n is 18
  - Note that you cannot write `n=3a`
- `n=a/2` Now n is 3
- `n=a%4` Now n is 2
- `n=a**2` Now n is 36 (6^2)

Python - Order of Operations

Python knows the order of operations that you use in maths, including brackets

Examples

`a=11`

- `n=3+a*2` n is now 25
- `n=(3+a)*2` n is now 28
    - Note that you cannot use square brackets [ ] here.

Simple comparison with if

Start a new file. Name it `simple_comparison.py`

Add the following code (and run it)

```
# a simple comparison with if
first_number=5
second_number=first_number+7
print(first_number)
print(second_number)
```

Your output should be

```
5
12
```

Increment a variable

Add the following code at the end of `simple_comparison.py` and run it.

```
# increase first_number by 7
first_number+=7
print(first_number)
```

Your output should now look like this

```
5
12
12
```

The code `first_number+=7` means:

Increase `first_number` by 7.

It is the same as `first_number=first_number+7`

Compare numbers

Add the following code and run it.

```
# compare the numbers
if first_number==second_number:
      print('The numbers are equal')
```

Output

```
5
12
12
The numbers are equal
```

Explanation on next slide

Compare numbers – Explanation

```
if first_number==second_number:
      print('The numbers are equal')
```

The code means:

- Check if `first_number` equals `second_number`
- You must use `==` for comparison (not `=`)
- The colon `:` at the end means do the indented steps that follows if it is True
- The print action must be indented (use tab)
- If it is not True the code does nothing

More Comparisons

Add the following code and run it.

```
# compare the first number with a new number
new_number=5
print(first_number)
if first_number>new_number:
      print('is greater than')
print(new_number)
```

Output

```
12
is greater than
5
```

Printing on One Line

Adjust your print statements as follows.

```
print(first_number, end='')
print(' is greater than ', end='')
print(new_number)
```

Output

```
12 is greater than 5
```

The code , `end =''` means

Don't print a line break at the end

`Note: you must include the , (comma)`

Comparison with else

Add more code to look like this (and run it)

```
# compare the first number with a new number
new_number=5
print()
print(first_number, end='')
if first_number>new_number:
      print(' is greater than ', end='')
else:
      print(' is not greater than ', end='')
print(new_number)
```

print() prints a blank line

Output

```

12 is greater than 5
```

Explanation on next slide

Explanation of Else

```
if first_number>new_number:
      print(' is greater than ', end='')
else:
      print(' is not greater than ', end='')
```

The code means:
Check if first_number is bigger than new_number
If it is True, print `' is greater than '`
The else part is what to do if it is not True
Then we print `' is not greater than '`
Note that else must be followed by a colon : and the code below it must be indented

Change new_number to get a different result

Change one line of your code to look like this (and run it)

```
new_number=25
```

Output

```
12 is not greater than 25
```

Python Assignment Operators

- `=`
- `+=`
- `-=`

etc

Examples

- `n=3` Now n is 3.
- `n+=1` Now n is 4. Same as `n=n+1`
- `n-=6` Now n is -2. Same as `n=n-6`

Python Comparison Operators

- == (equal) Note: you cannot use = to compare
- != (not equal)
- > (greater than)
- < (less than)
- >= (greater than or equal)
- <= (less than or equal)